**Image Classification using Convolutional Neural Networks**

**Udacity: Machine Learning Engineering Nanodegree - Capstone Project**

**Bryce Simmer**

**8/17/2017**

**I. Definition**

**Project Overview**

The trend in technology today is to build systems and machines that can anticipate our needs and satisfy our desires in a responsive and accurate manner. In order to achieve this idealized interactivity with our technology we need to not only increase the computational power of our systems, but we need to gather and analyze user feedback in new ways. Computer vision, or the ability of computers to gain high-level understanding of digital images or videos, is crucial to narrowing the divide between humans and their machines. Autonomous vehicles, robotics, and facial recognition software are just a few technologies that have benefited from improved computer vision. Computer vision is also one of the most important applications of deep learning, which is the focus of this project. We will be using a Convolutional Neural Network (CNN) to classify the images in the CIFAR-10 dataset and analyzing the results.

**Problem Statement**

****CNNs are very similar to an ordinary neural network: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer. One main difference between CNN architectures and ordinary neural networks is CNNs explicitly assume that the inputs are images. The layers of a CNN have neurons arranged in 3 dimensions, allowing for a vast reduction in the amount of parameters in the network. Three main types of layers are used to build CNN architectures: Convolution Layer, Pooling Layer, and Fully-Connected Layer. [3]

Data flow graphs describe mathematical computation with a directed graph of nodes & edges. Nodes typically implement mathematical operations, but can also represent endpoints to feed in data, push out results, or read/write persistent variables. Edges describe the input/output relationships between nodes. These data edges carry dynamically-sized multidimensional data arrays, or tensors. The flow of tensors through the graph is where TensorFlow gets its name. Nodes are assigned to computational devices and execute asynchronously and in parallel once all the tensors on their incoming edges becomes available.

**Metrics**

A confusion matrix contains information on the actual and predicted classifications performed by a system. In the example below numbers along the leading diagonal of the table represent digits that have been classified correctly, while off-diagonal values show the number of misclassifications. Hence, small numbers along the leading diagonal show cases in which classification performance has been poor, as with ‘8’ in the table. Here, the actual digit ‘0’ has been mis-classified as ‘8’ ten times and as ‘6’ once, while the digit ‘1’ mis-classified as ‘7’ six times. Conversely, the digit ‘2’ has never been mis-classified.

Predictions are evaluated using the multi-class logarithmic loss. Each image has been labeled with one true class. For each image, you must submit a set of predicted probabilities (one for every image). The formula is then, where *N* is the number of images in the test set, *M* is the number of image class labels, *log* is the natural logarithm, *yij* is 1 if observation *i* belongs to class *j* and 0 otherwise, and *pij* is the predicted probability that observation *i* belongs to class *j*. The probabilities for a given image are not required to sum to one because they are rescaled prior to being scored (each row is divided by the row sum). In order to avoid the extremes of the log function, predicted probabilities are replaced with *max(min(p, 1 – 10-15), 10-15)*.

****

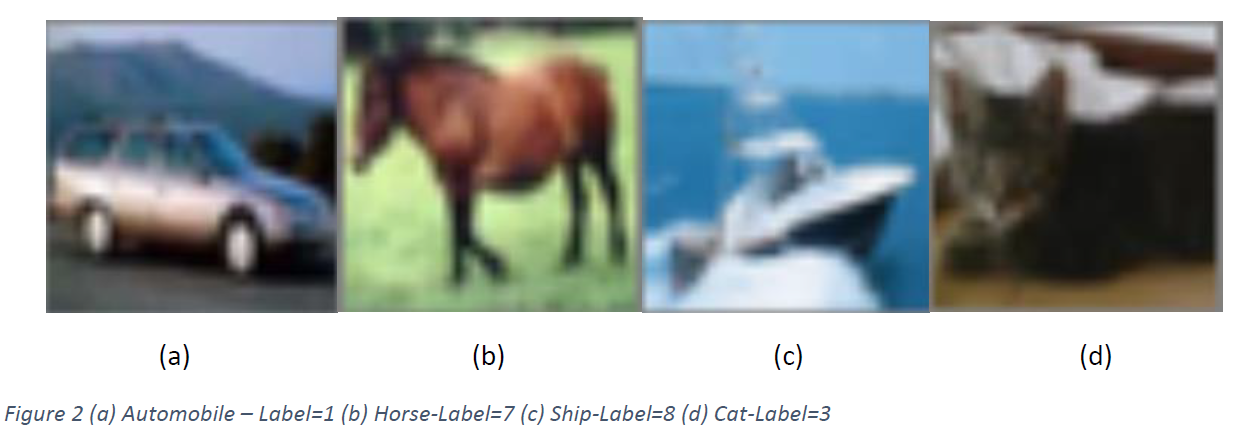
**II. Analysis**

**Data Exploration**

In this study, we have used the publicly available image dataset of CIFAR-10 [1]. There are 60,000 total images in this dataset. 50,000 images are categorized for training and the remaining 10,000 are categorized for testing. There are 10 possible image classes: Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, and Truck. There are 10 training batches, containing the images in random order. Some batches may contain more images from one class than another. Each batch contains 5,000 images from each class.

The images are 32x32x3 RGB format (width=height=32 and one dimension for each color component R, G, B). Figure 2 illustrates some of the images and their labels from the Cifar-10 dataset.

We have converted the category information (labels) to one-hot encoding that is used by the final classification. For instance, we convert label=3 to a vector as [0 0 0 1 0 0 0 0 0 0] (labels starts from 0 to 9). We use this vector format in the classification to calculate training error at every step of the training.

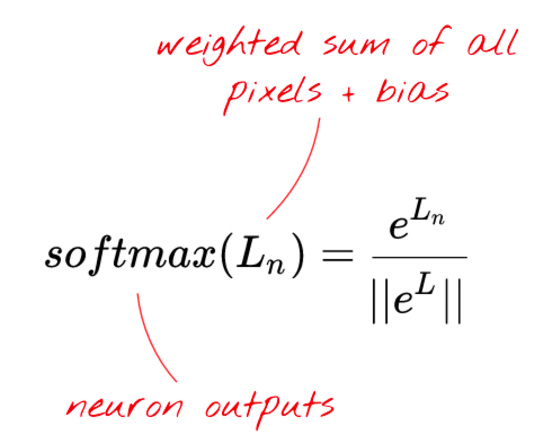
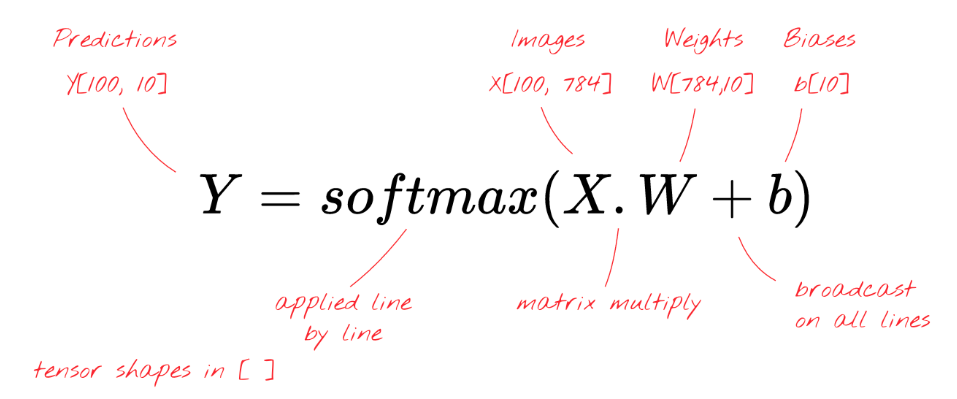


**Exploratory Visualization**

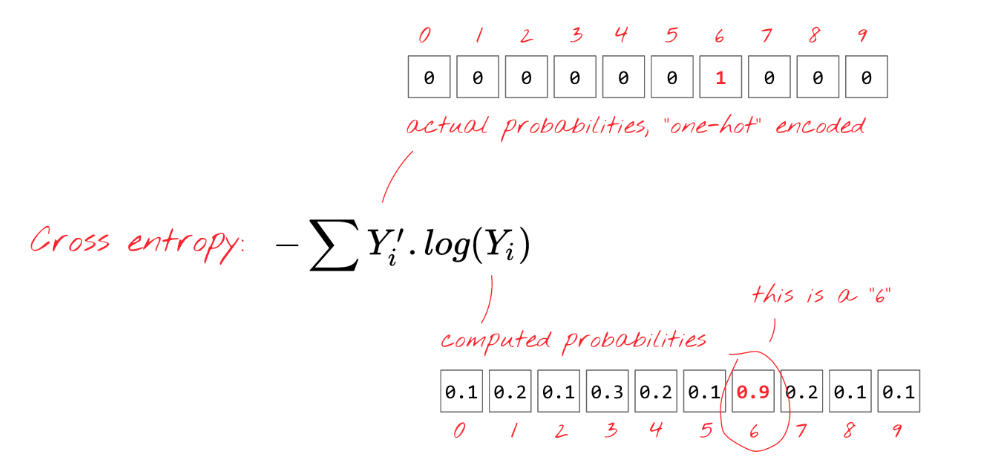
**Algorithms and Techniques**

For a classification problem, an activation function that works well is softmax. Applying softmax on a vector is done by taking the exponential of each element and then normalizing the vector (using any norm, for example the ordinary Euclidean length of the vector).

Why is "softmax" called softmax ? The exponential is a steeply increasing function. It will increase differences between the elements of the vector. It also quickly produces large values. Then, as you normalise the vector, the largest element, which dominates the norm, will be normalised to a value close to 1 while all the other elements will end up divided by a large value and normalised to something close to 0. The resulting vector clearly shows which was its largest element, the "max", but retains the original relative order of its values, hence the "soft".

"**Broadcasting**" is a standard trick used in Python and numpy, its scientific computation library. It extends how normal operations work on matrices with incompatible dimensions. "Broadcasting add" means "if you are adding two matrices but you cannot because their dimensions are not compatible, try to replicate the small one as much as needed to make it work."



"**Learning rate**": you cannot update your weights and biases by the whole length of the gradient at each iteration. It would be like trying to get to the bottom of a valley while wearing seven-league boots. You would be jumping from one side of the valley to the other. To get to the bottom, you need to do smaller steps, i.e. use only a fraction of the gradient, typically in the 1/1000th region. We call this fraction the "learning rate".

Why work with "**mini-batches**" of 100 images and labels ?

You can definitely compute your gradient on just one example image and update the weights and biases immediately (it's called "stochastic gradient descent" in scientific literature). Doing so on 100 examples gives a gradient that better represents the constraints imposed by different example images and is therefore likely to converge towards the solution faster. The size of the mini-batch is an adjustable parameter though. There is another, more technical reason: working with batches also means working with larger matrices and these are usually easier to optimise on GPUs.

**Benchmark**

For benchmarking purposes I will be using a Receiver Operating Characteristic (ROC) curve, a Detection Error Trade-Off (DET) curve, and confusion matrix to assess the performance characterization of my computer vision solution. A ROC curve is a plot of false positive rate against true positive rate as some parameter is varied. The closer the curve approaches the top left-hand corner of the plot, the more accurate the test. The plot highlights the trade-off between the true positive rate the false-positive rate.



A DET curve is a plot of false positive rate versus false negative rate and thus gives equal emphasis to both types of error. The plot usually has logarithmic scales on both axes, so DET curves tend to be more spread out than ROC curves, making it easier to distinguish individual algorithms’ results. DET curves can be used to plot matching error rates and decision error rates as well as confidence intervals/boxes.

****

**III. Methodology**

**Data Preprocessing**

**Implementation**

**Refinement**

**IV. Results**

**Model Evaluation and Validation**

**Justification**

**V. Conclusion**

**Free-Form Visualization**

**Reflection**

**Improvement**

**References**

[1] “Learning Multiple Layers of Features from Tiny Images” <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>

[2] “Tensorflow” <https://www.tensorflow.org/>

[3] "CS231n Convolutional Neural Networks for Visual Recognition" <http://cs231n.github.io/convolutional-networks/>

[4] “Learn TensorFlow and Deep Learning, without a Ph.D.” <https://cloud.google.com/blog/big-data/2017/01/learn-tensorflow-and-deep-learning-without-a-phd>

[5] “Why You Should Use Cross-Entropy Error Instead of Classification Error or Mean Squared Error for Neural Network Classifier Training” <https://jamesmccaffrey.wordpress.com/2013/11/05/why-you-should-use-cross-entropy-error-instead-of-classification-error-or-mean-squared-error-for-neural-network-classifier-training/>

[] “Distinctive Image Features from Scale-Invariant Keypoints” <https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf>

[] http://www.vision.ee.ethz.ch/en/publications/papers/articles/eth\_biwi\_00517.pdf

[] https://computervisionblog.wordpress.com/2012/01/29/how-things-work-amazon-flow-appalgorithm/

[] http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neuralnetworks.

pdf

[] https://en.wikipedia.org/wiki/Gradient\_descent

[] https://en.wikipedia.org/wiki/Stochastic\_gradient\_descent

[] <https://en.wikipedia.org/wiki/Backpropagation>

[1] “The CIFAR-10 dataset” <https://www.cs.toronto.edu/~kriz/cifar.html>